

IPRs and their data analysis

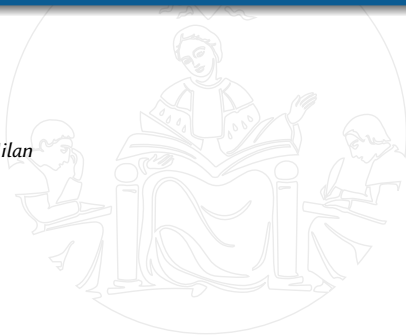
MODULE 5 – SQL basics

Jacopo Staccioli, PHD^{†‡}

[†] *Università Cattolica del Sacro Cuore, Milan*

[‡] *Scuola Superiore Sant'Anna, Pisa*

a.y. 2024-25



1 SQL basics

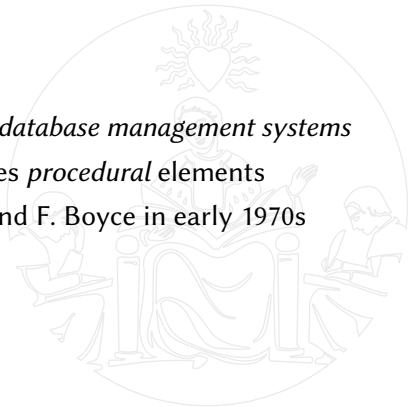
- What is SQL
- SQL syntax
- Data definition
- Data manipulation
- Data query
- Operators and data types



What is SQL

SQL : “*Structured Query Language*”

- pronounced /,ɛs,kju:ˈɛl/ or /ˈsi:kwəl/
- preferred language for managing data held in *relational database management systems*
- essentially a *declarative* language, although often includes *procedural* elements
- developed at IBM by Donald D. Chamberlin and Raymond F. Boyce in early 1970s
- ANSI standard since 1986 and ISO standard since 1987



What is SQL (cont'd)

- SQL can be informally partitioned as 4 inner *sublanguages*

DDL : *data definition language*

- for creating and modifying database objects (e.g. tables, indices)

DML : *data manipulation language*

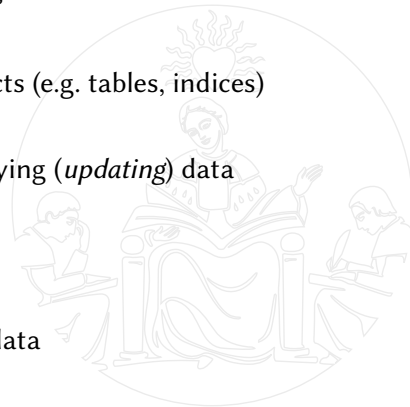
- for adding (*inserting*), deleting, and modifying (*updating*) data

DQL : *data query language*

- for performing data retrieval queries

DCL : *data control language*

- for authorising and controlling access to data



What is SQL (cont'd)

- we focus especially on **CRUD** operations

Create : INSERT

Read : SELECT

Update : UPDATE

Delete : DELETE

- $\text{CRUD} \subseteq (\text{DML} \cup \text{DQL})$

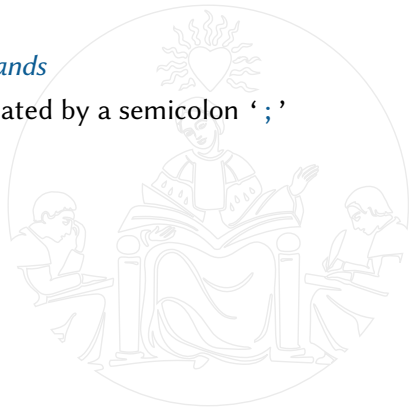


1 SQL basics

- What is SQL
- SQL syntax
- Data definition
- Data manipulation
- Data query
- Operators and data types



- we focus on the MariaDB ($\approx$ MySQL) syntax¹
- SQL input consists of a sequence of *statements* or *commands*
- a statement is composed of a sequence of *tokens*, terminated by a semicolon ‘;’
- a token can be
 - a *keyword* e.g. ‘SELECT’
 - an *identifier* e.g. the name of a table or attribute
 - a *literal* e.g. ‘2019’ (*constant*)



¹<https://mariadb.com/kb/en/sql-statements/>

SQL clauses

- data definition language (DDL)
 - CREATE : defines a new table
 - ALTER : changes the types of records in a table
 - DROP : removes a table
- data manipulation language (DML)
 - INSERT : adds records to a table
 - UPDATE : changes part of a record
 - DELETE : removes records from a table
- data query language (DQL)
 - SELECT : retrieves data
- data control language (DCL)
 - GRANT : gives database permissions
 - REVOKE : removes database permissions



1 SQL basics

- What is SQL
- SQL syntax
- **Data definition**
- Data manipulation
- Data query
- Operators and data types



Data definition

■ create and delete a table

SYNTAX

```
CREATE TABLE <table> (  
    <column1> <datatype> [<constraint>],  
    <column2> <datatype> [<constraint>],  
    <column3> <datatype> [<constraint>],  
    <column4> <datatype> [<constraint>],  
    ...  
);
```

```
DROP TABLE [IF EXISTS] <table>;
```

EXAMPLE

```
CREATE TABLE patents (  
    id INTEGER PRIMARY KEY,  
    auth CHAR(2),  
    num TEXT NOT NULL,  
    kind VARCHAR(2) DEFAULT 'A1',  
    year NUMERIC(4,0) CHECK (year > 1800),  
    CHECK (year < 3000)  
);
```

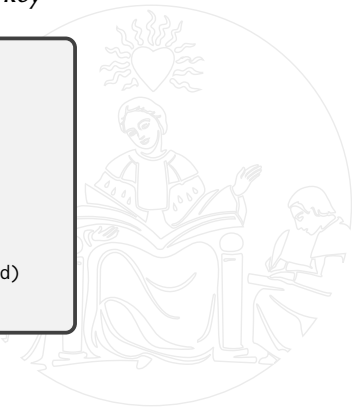
```
DROP TABLE IF EXISTS patents;
```



Data definition (cont'd)

- create a second table and cross-reference the first via *foreign key*

```
CREATE TABLE inventors (  
    id INTEGER PRIMARY KEY,  
    name TEXT,  
    surname TEXT,  
    sex CHAR(1),  
    address TEXT,  
    city TEXT,  
    country CHAR(2),  
    patent_id INTEGER REFERENCES patents (id)  
);
```



Data definition (cont'd)

patents

id	auth	num	kind	year

primary key

foreign key

inventors

id	name	surname	sex	address	city	country	patent_id



Data definition (cont'd)

- add/remove a column to/from a table
- rename/change type of a column in a table

```
ALTER TABLE <table>  
ADD COLUMN <column> <datatype>;
```

```
ALTER TABLE patents  
ADD COLUMN app_year NUMERIC(4)  
CHECK (app_year <= year);
```

```
ALTER TABLE <table>  
RENAME COLUMN <column> TO <newcolumn>;  
ALTER TABLE <table>  
ALTER COLUMN <column> TYPE <datatype>;
```

```
ALTER TABLE patents  
RENAME COLUMN app_year TO app_date;  
ALTER TABLE patents  
ALTER COLUMN app_date TYPE DATE;
```

```
ALTER TABLE <table> DROP COLUMN <column>;
```

```
ALTER TABLE patents DROP COLUMN app_date;
```



1 SQL basics

- What is SQL
- SQL syntax
- Data definition
- **Data manipulation**
- Data query
- Operators and data types



Data manipulation

■ insert rows in a table

```
INSERT INTO <table> VALUES
(<value1>, <value2>, ...),
(<value1>, <value2>, ...),
...
;
```

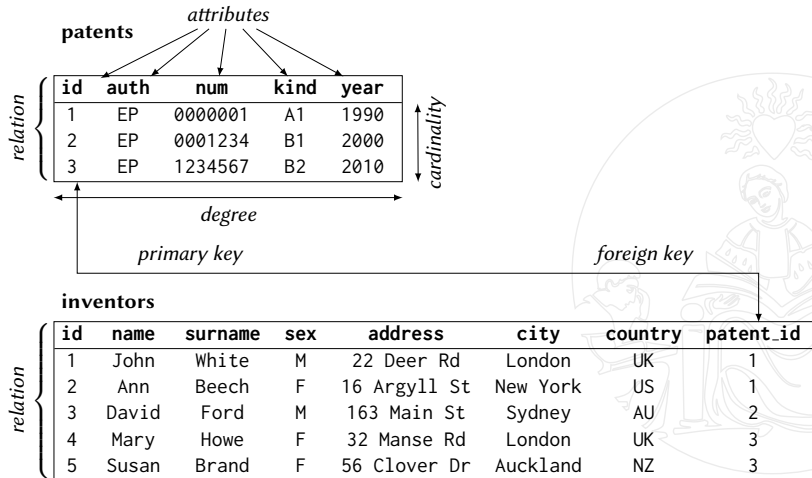
```
INSERT INTO patents VALUES
(1, 'EP', '0000001', 'A1', 1990),
(2, 'EP', '0001234', 'B1', 2000),
(3, 'EP', '1234567', 'B2', 2010)
;
```

```
INSERT INTO <table> (<col1>, <col2>, ...) VALUES
(<value1>, <value2>, ...),
(<value1>, <value2>, ...),
(<value1>, <value2>, ...),
(<value1>, <value2>, ...),
...
;
```

```
INSERT INTO inventors (id, patent_id, ...) VALUES
(1, 1, 'John', 'White', 'M', '22 Deer St', ...),
(2, 1, 'Ann', 'Beech', 'F', '16 Argyll St', ...),
(3, 2, 'David', 'Ford', 'M', '163 Main St', ...),
(4, 3, 'Mary', 'Howe', 'F', '32 Manse Rd', ...),
(5, 3, 'Susan', 'Brand', 'F', '56 Clover Dr', ...)
;
```



Data manipulation (cont'd)



Data manipulation (cont'd)

■ update rows in a table

```
UPDATE <table>  
SET <expression>;
```

```
UPDATE <table>  
SET <expression>  
WHERE <expression>;
```

```
UPDATE patents  
SET year = year + 1;
```

```
UPDATE patents  
SET year = 2000  
WHERE year = 1990;
```

Data manipulation (cont'd)

■ delete rows from a table

```
DELETE FROM <table>;
```

```
DELETE FROM <table>  
WHERE <expression>;
```

```
DELETE FROM patents;
```

```
DELETE FROM patents  
WHERE year <= 2000;
```



1 SQL basics

- What is SQL
- SQL syntax
- Data definition
- Data manipulation
- Data query
- Operators and data types



Data query

■ retrieve rows from a table

```
SELECT * FROM <table>;
```

```
SELECT * FROM patents;
```

```
SELECT <column>  
[AS <newname>]  
FROM <table>;
```

```
SELECT auth  
AS pto  
FROM patents;
```

```
SELECT CONCAT(<col1>, <col2>, <col3>)  
AS <newname>  
FROM <table>;
```

```
SELECT CONCAT(auth, num, kind)  
AS publication  
FROM patents;
```

N.B. SQLite does not support the CONCAT() function and requires a double pipe || to concatenate strings



Data query (cont'd)

- retrieve rows from a table based on condition(s) and sort/group by columns

```
SELECT * FROM <table>  
[WHERE <condition>]  
[ORDER BY <col1>, <col2>, ...] [DESC]  
[LIMIT <number>];
```

```
SELECT * FROM inventors  
WHERE sex = 'F'  
ORDER BY surname  
LIMIT 2;
```

```
SELECT <col1>, <col2>, ..., COUNT(*) AS <name>  
FROM <table>  
GROUP BY <col1>, <col2>, ... ;
```

```
SELECT sex, COUNT(*) AS count  
FROM inventors  
GROUP BY sex;
```



Data query (cont'd)

- combine columns from multiple tables

```
SELECT * FROM <table1>  
[INNER] JOIN <table2>  
ON <condition>;
```

```
SELECT * FROM patents  
INNER JOIN inventors  
ON patents.id = inventors.patent_id;
```

id	auth	num	kind	year	id	name	surname	sex	address	city	country	patent_id
1	EP	0000001	A1	1990	1	John	White	M	22 Deer Rd	London	UK	1
1	EP	0000001	A1	1990	2	Ann	Beech	F	16 Argyll St	New York	US	1
2	EP	0001234	B1	2000	3	David	Ford	M	163 Main St	Sydney	AU	2
3	EP	1234567	B2	2010	4	Mary	Howe	F	32 Manse Rd	London	UK	3
3	EP	1234567	B2	2010	5	Susan	Brand	F	56 Clover Dr	Auckland	NZ	3

INNER JOIN: returns only records whose ON.<attribute> exists in both tables

LEFT JOIN: returns all records from the first table

RIGHT JOIN: returns all records from the second table

Create table on the fly

- query results can be saved as separate tables
- useful not to repeat complex and time consuming queries (e.g. with multiple JOINS)

```
CREATE TABLE <owndatabase>.<owntable> AS  
SELECT * FROM <database>.<table>  
... ;
```

- requires “write” privileges on <owndatabase>
- be sure to specify the database name if CREATE is on a different schema
 - or if it is not implied by the database connection



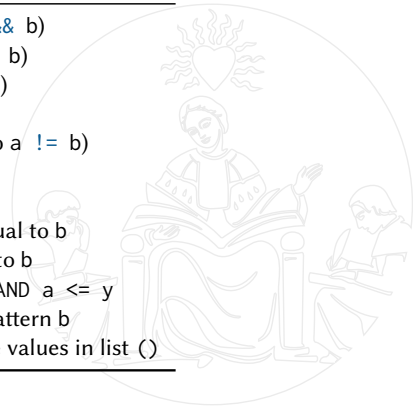
1 SQL basics

- What is SQL
- SQL syntax
- Data definition
- Data manipulation
- Data query
- Operators and data types



Operators

Operator	Description
a AND b	boolean AND (also a && b)
a OR b	boolean OR (also a b)
NOT a	boolean NOT (also ! a)
a = b	a is equal to b
a <> b	a is not equal to b (also a != b)
a > b	a is greater than b
a < b	a is less than b
a >= b	a is greater than or equal to b
a <= b	a is less than or equal to b
a BETWEEN x AND y	equivalent to a >= x AND a <= y
a LIKE b	a contains character pattern b
a IN ()	a is equal to any of the values in list ()



Data types – numeric

Name	Size	Description	Range
TINYINT	1 bytes	small-range integer	−128 to +127
SMALLINT	2 bytes	small-range integer	−32768 to +32767
MEDIUMINT	3 bytes	small-range integer	−8388608 to +8388607
INT[EGER]	4 bytes	typical choice for integer	−2147483648 to +2147483647
BIGINT	8 bytes	large-range integer	−9223372036854775808 to +9223372036854775807
NUMERIC(M,D)	variable	user-specified precision, exact	$1 \leq M \leq 65$; $0 \leq D \leq M \leq 30$
FLOAT(P)	4 bytes	variable-precision, inexact	$0 \leq P \leq 23$
DOUBLE(P)	8 bytes	variable-precision, inexact	$24 \leq P \leq 53$
YEAR	1 byte	‘yyyy’	‘1901’ to ‘2155’
DATE	3 bytes	‘yyyy-mm-dd’	‘1000-01-01’ to ‘9999-12-31’
TIME	3 bytes	‘hh:mm:ss’	‘−838:59:59’ to ‘838:59:59’
DATETIME	8 bytes	‘YYYY-MM-DD hh:mm:ss’	‘1000-01-01 00:00:00’ to ‘9999-12-31 23:59:59’
TIMESTAMP	4 bytes	‘YYYY-MM-DD hh:mm:ss’	‘1970-01-01 00:00:00’ to ‘2038-01-19 03:14:07’



Data types – literal

Name	Size	Description	Range
CHAR(M)	M bytes	fixed-length (M) string	$0 \leq M \leq 255$
VARCHAR(M)	1 or 2 + L bytes	variable-length ($L \leq M$) string	$0 \leq M \leq 65535$
TINYTEXT	1 + L bytes	variable-length string	$0 \leq L \leq 255$
TEXT	2 + L bytes	variable-length string	$0 \leq L \leq 65535$
MEDIUMTEXT	3 + L bytes	variable-length string	$0 \leq L \leq 16777215$
LONGTEXT	4 + L bytes	variable-length string	$0 \leq L \leq 4294967295$

- literal data types can be indexed
 - *TEXT types can be indexed up to a fixed specified length
- with a normal (BTREE) index
 - left-aligned queries (e.g. LIKE '<WORD>%') exploit the index
 - other *wildcard* queries (e.g. LIKE '%<WORD>') are inefficient
- with a **FULLTEXT** index, all textual queries exploit the index



Housekeeping

- DBeaver Community Edition
 - SQL client with intuitive GUI
 - free and open source software
 - cross-platform
 - supports many popular RDBMSs
 - connects via JDBC API
 - <https://dbeaver.io/>
- we will use it for querying PATSTAT, PATENTSVIEW...
- you can already use it to practice SQL
 - find the chinook.db SQLite example database in /home/<USER>/data/
 - <https://sqlite.org/docs.html>

